CMPSC-F265 Final Exam
Sample Practice

NAME:	Student ID:

Rules:

- 150 minutes, 100 possible points
- Closed book/notes. One A4 size one-sided cheat sheet
- No calculators, no electronic devices

Problem	Score
1 (Time Complexity Analysis)	
2 (Stack/Queue/Priority Queue)	
3 (Sorting Algorithm)	
4 (Linked List)	
5 (Binary Tree)	
6 (Binary Search Tree)	
7 (Binary Heap and Heap Sort)	
8 (Hash table)	
9 (Graph)	
10. (Others)	
Total (100)	

Problem 1 Complexity Analysis and Big O Notation

1. Let *n* be the size of input, given the number of steps for different algorithms in the following table, write down the corresponding big O complexity.

Number of steps	Complexity order (big O)
$n * \log(\frac{n}{2}) + \sqrt{n} * \sqrt{n}$	O(nlogn) + O(n) = O(nlogn)
$(n+1)^2 + 2^{n-1}$	O(n^2) + O(2^n) = O(2^n)
$\log(4 * n) + \sqrt{n/4}$	O(logn) + O(sqrt(n)) = O(sqrt(n)) // You will not have sqrt(n) tested. Just for your refenrece: O(logn) < O(sqrt(n))

[2pts] Determine the complexity order (big O) for each of the following codes

```
for (int i=n/2; i< (n/2 + 2); i++)
{
     //some statement
}
```

O(1)

```
for (int i=1; i<n; i=i*2)
{
     for (int j=n; j>1; j--)
     {
          //some statement
     }
}
```

logn * n, so O(nlogn)

```
for (int i=1; i<n; i+=2)
 for (int  j=1; j<2*n; j++)
    for (int k=1; k<n; k++) {
          // some statement
    }
 }
}
```

n * 2*n * n, so O(n^3)

Problem 2 Stack, Queue, and Priority Queue

1. Explain in English what the effect of the following mystery method is:

```
void mystery(Queue<T> q) {
     Stack<T> s = new Stack<T>();
     while (!q.isEmpty()) [
          T e = q.remove();
          s.push(e);
     }
     while (!s.isEmpty()) {
          T e = s.pop();
          q.add(e);
     }
```

It will reverse the original elements in Queue q.

2. Suppose that a minus sign in the input indicates *dequeue* the priority queue, and other string indicates *enqueue* the string into a **queue**. What will be the content (from front to rear) of the queue with the following inputs. Suppose the priority of the strings is determined by their natural order (i.e. 'A'<"B' and 'B' < 'C')

B D - A E - - F C - - M P - O G Z K - -

O G Z K

3. Consider the **sorted-array-based** implementation of a **Priority Queue**. Implement the **enqueue()** method.

- This would be a Minimum Priority Queue, in the sense that the internal array will be sorted in descending order, and that each time when dequeuing an element, the last element in the array will be returned; To enqueue an element, you need to find the proper place to insert it.
- You may also assume that the internal array is very large in size so that you do not need to consider the situation when the number of elements has reached the full capacity of the array.

```
class PriorityQueue
  {
  // array in sorted order, from min at 0 to max at size-1
  private int maxSize;
  private int[] queArray;
  private int nItems;
//--------------------------------------------------------------
  public PriorityQueue(int max)          // constructor
    {
```

```
      maxSize = max;
      queArray = new int[maxSize];
      nItems = 0;
      }
//--------------------------------------------------------------
   public void enqueue(int item)    // enqueue an element
   {
     // YOUR CODES

// pay attention need to shift some elements to the right.
int i=nItems-1;
while (i>=0 && queArray[i]<item){
    queArray[i+1] = queArray[i];
     i--;
}
queArray[i+1]=item;
}

   public int dequeue()            // dequeue the minimum element
   { return queArray[--nItems];

   }
}
```

4. Suppose you were asked to write a method that accepts a string as argument, and remove the adjacent duplicate characters in the string. **Please implement this method using a Stack**. For example:
removeDuplicate("abddbc") will return "ac"
removeDuplicate("abba") will return an empty string.

```
public static String removeDuplicate(String s){
   // YOUR CODES
   Stack<Character> theStack = new Stack<Character>();
   for (int i=0; i<s.length(); i++) {
     char c = s.charAt(i);
     if (!theStack.isEmpty() && theStack.peek()==c){
       theStack.pop();
     }
     else {
       theStack.push(c);
     }
   }

   String newS = "";
   while (!theStack.isEmpty()) {
     newS = theStack.pop() + newS;
   }
   return newS;
 }
```

Problem 3. Sorting Algorithms

1. Given the following array:

7 0 5 4 3 6 6 9

a) Show the content of the array for the first two iterations using Seletion sort to sort it.

0 7 5 4 3 6 6 9
0 3 5 4 7 6 6 9

b) Show the content of the array for the first two iterations using Insertion sort to sort it.

0 7 5 4 3 6 6 9
0 5 7 4 3 6 6 9

c) Show the content of the array for the first two iterations using Bubble sort to sort it.

After the first iteration: 0 5 4 3 6 6 7 9
After the second iteration: 0 4 3 5 6 6 7 9

d) Show the content of the array after the first partition using Quick sort (suppose you choose the first element as Pivot).

6 0 5 4 3 6 7 9

2. Suppose the running time of applying Selection Sort on a given array is 1024ms. How much time do we need to spend using Merge sort to sort on the same array?

Let us consider the average case.
For selection sort, it is O(n^2): n^2 = 2^10 => n = 2^5 = 32
Merge Sort: O(nlogn) => 32 * log2^5 = 32 * 5 = 160 ms

3. Given the following unsorted array, what is the best pivot you can choose to do quick sort?

10	5	20	4	15	1	7	6	9

The best pivot would be the median value of all the elements.

How to find the median:
1) firstly sort the array: 1 4 5 6 7 9 10 15 20
2) There are totally 9 elements in the array, so the median would be the (8-0)/2 = 4, the one with index 4, which is 7.

So we should choose 7 to work as the pivot.

Why choose median as the pivot, this is to make sure that the left subarray and the right sub-array would be of equivalent size.

4. Complete the following table with big O time complexities of sorting algorithms.

Algorithm	Worst-case time complexity	Average-case time complexity
Insertion sort	O(n^2)	O(n^2)
Merge sort	O(nlogn)	O(nlogn)
Quick sort (pivot being the first item in the array)	O(nlogn)	O(nlogn)
Heap sort	O(nlogn)	O(nlogn)

Problem 4. Linked List

1. Fill the following table with time complexities of different operations in the specified data structures.

	insertFirst()	**insertLast()**	**deleteLast()**
Singly linked list	O(1)	O(n)	O(n)
Double-ended singly linked list	O(1)	O(1)	O(n)
Double-ended doubly linked list	O(1)	O(1)	O(1)

2. Which operation is O(n) in a double-ended singly linked list, but O(1) in a double-ended doubly linked list?

   Remove the last element.

3. Assume that LL is a SINGLY linked list with the head node known and one other internal node M known that is not the last node. Write few lines of code to accomplish the following. You may assume that each node has a *next* pointer. There is no other existing method for you to call.

a). Find and Return the one-to-the-last link in a single-end singly linked list.

```
Node current = head;
Node previous = head;

while (current.next!=null){
        previous = current;
        current = current.next;
}
return previous;
```

b) Delete the head node

```
head = head.next;
```

c). Swap head and the M node (you are not allowed to simply swap the data)

```
Node temp1= head;
Node temp2 = head.next;
Node temp3 = head;

while (temp1!=M){
    temp3 = temp1;
```

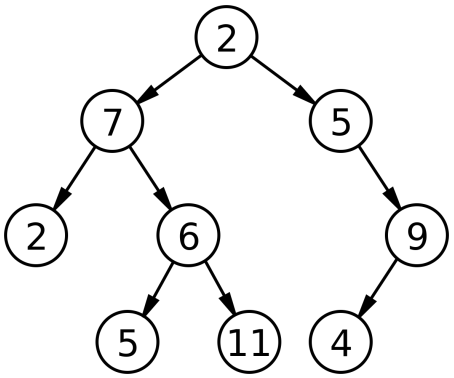
```
    temp1= temp1.next;
}
temp1 = temp1.next;
// so temp3 is referencing the previous Node of M;  temp1 is referencing the next Node
// of M.

M.next = temp2;
temp3.next = head;
head.next = temp1;
head = M;
```

3. Complete the following method in the single-ended singly LinkedList class. The method removeN(int n) returns nothing, but it will remove the Nth node in the linked list from the front. You have no other methods to call.

```
public void removeN(int n){
// YOUR CODES
    if (n==1) {
        first = first.next;
        return;
    }
    Link current = first;
    Link previous = first;
    int count = 1;
    while (current!=null && count<n){
        previous = current;
        current = current.next;
        count++;
    }
    previous.next = current.next;
}
```

Problem 5. Binary Tree

a). Given the following binary tree:



(1) Is this tree a complete binary tree? (Y/N) ___N_____

(2) What is the height of the tree? ___3_____

(3) What is the depth of node "9" ____2_____

(4) Show the Pre-Order traversal sequence of numbers:

2 2 2 6 5 11 5 9 4

(5) Show the In-Order traversal sequence of numbers:

2 7 5 6 11 2 5 4 9

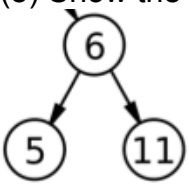
(6) Show the Post-Order traversal sequence of numbers:

2 5 6 11 7 4 9 5 2

(7) Show the level-Order traversal sequence of number:

2 7 5 2 6 9 5 11 4

(8) Show the right-subtree of node "7".

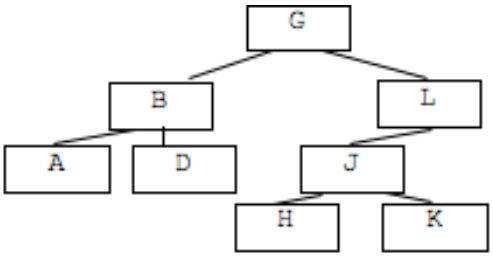


b). Consider the following class defining a Tree Node in a binary tree, and the following mystery method:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     String val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(String x) { val = x; }
 * }
 */
```

```
public void mystery(TreeNode n) {
      Stack<TreeNode> aStack = new Stack<TreeNode>();
      stk.push(n);
      while (!stk.isEmpty()) {
            n = aStack.pop();
            System.out.println(n.val);
            if (n.left != null)
                  aStack.push(n.left);
            if (n.right!= null)
                  aStack.push(n.right);
      }
}
```

Suppose that the above method is called with the root of the following tree as argument. Give the output of this method.



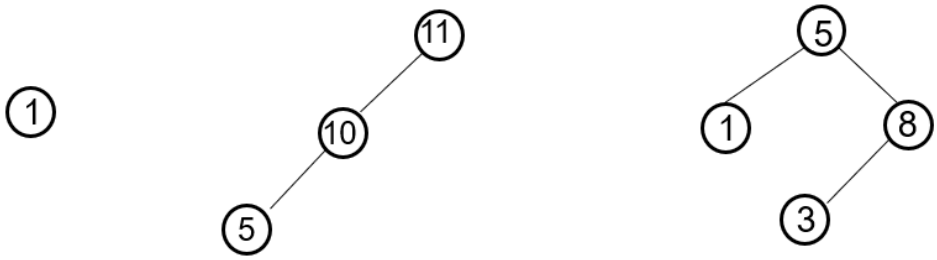
GLJKHBDA

c). Please finish implementing the following method which will return the sum of all the nodes' values in a binary tree

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */

public int getSum(TreeNode root){
        // YOUR CODES
  if (root==null) return 0;
  if (root.left==null && root.right==null) return root.val;
  return getSum(root.left) + getSum(root.right) + root.val;
}
```

Problem 6. Binary Search Tree

a). For each of the following, determine if it is a binary search tree or not.

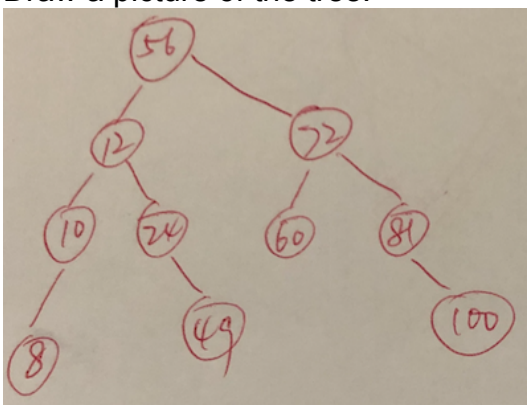


a is yes
b is yes
c is not, since node 3 is smaller than 5.

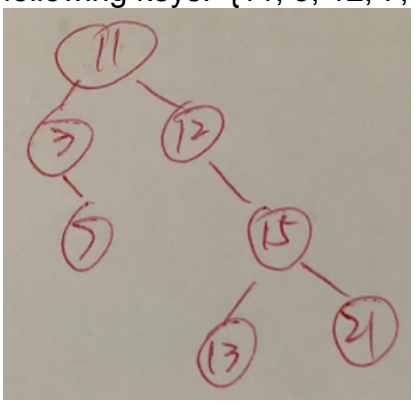
b). Suppose that we have a binary search tree whose keys are integers. A preorder traversal of this tree returns the following keys:

56 12 10 8 24 49 72 60 81 100

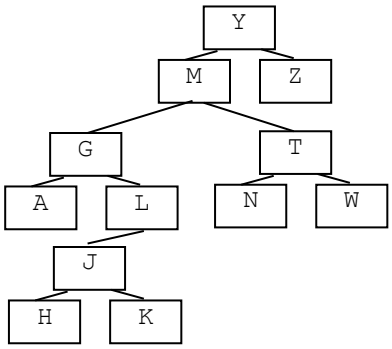
Draw a picture of the tree.



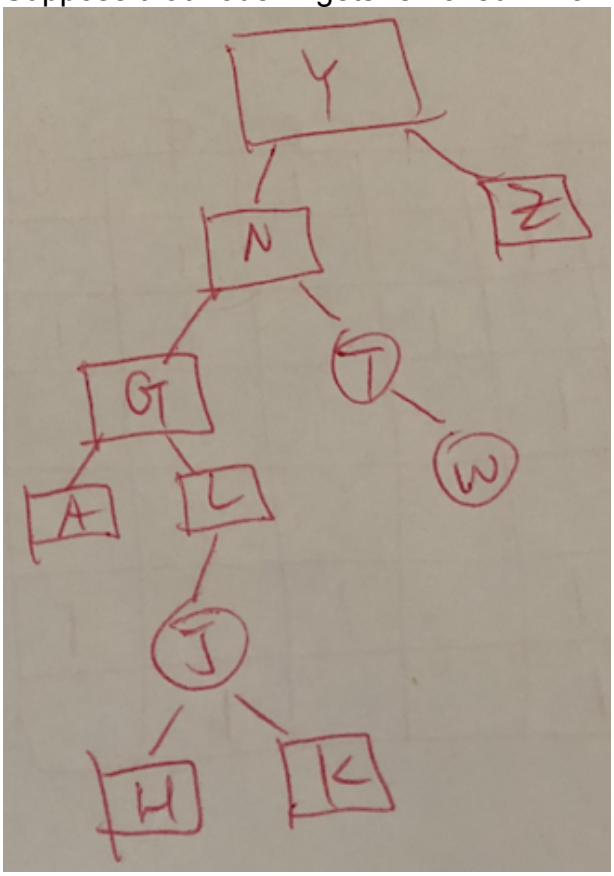
c). Suppose we want to build a binary search tree through successive insertions of the following keys: {11, 3, 12, 7, 15, 21, 13}. Draw the resulting binary search tree.



d). Consider the following binary search tree (note that only the key of each node is shown):



Suppose that node M gets removed. Draw a picture of the resulting binary search tree.



e). Consider the following class defineng a tree node in a binary search tree. Consider nodes n1 and n2 in a binary search tree. We say that n1 is an *ancestor* of n2 if there is a path (possibly of length 0) down the tree from n1 to n2. Write the code for the following method, which determines if node n1 is an ancestor of node n2.

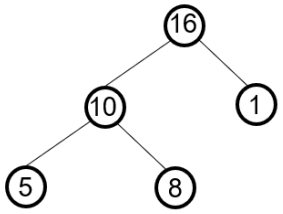
/**

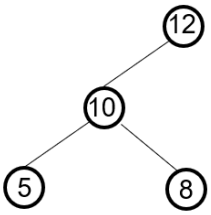
```
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */

public boolean isAncestor(TreeNode n1, TreeNode n2){
        // YOUR CODES
        if (n1==null) return false;
        else if (n1==n2) return true;
        else {
            if (n1.val<n2.val) return isAncestor(n1.right, n2);
            else return isAncestor(n1.left, n2);
        }
}

or iteratively:
while  (n1!=null) {
    if (n1==n2)  return true;
    if (n1.val<n2.val)  n1 = n1.right;
    else n2 = n2.left;
}
```

Problem 7. Binary Heap and Heap Sort
a). Is every binary heap balanced? Justify your answer briefly.

Yes. A binary heap is a complete binary tree, and according to the definition of complete binary tree, it is height balanced.

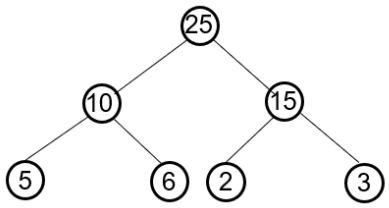
b). For each of the following, determine if it is a heap or not.

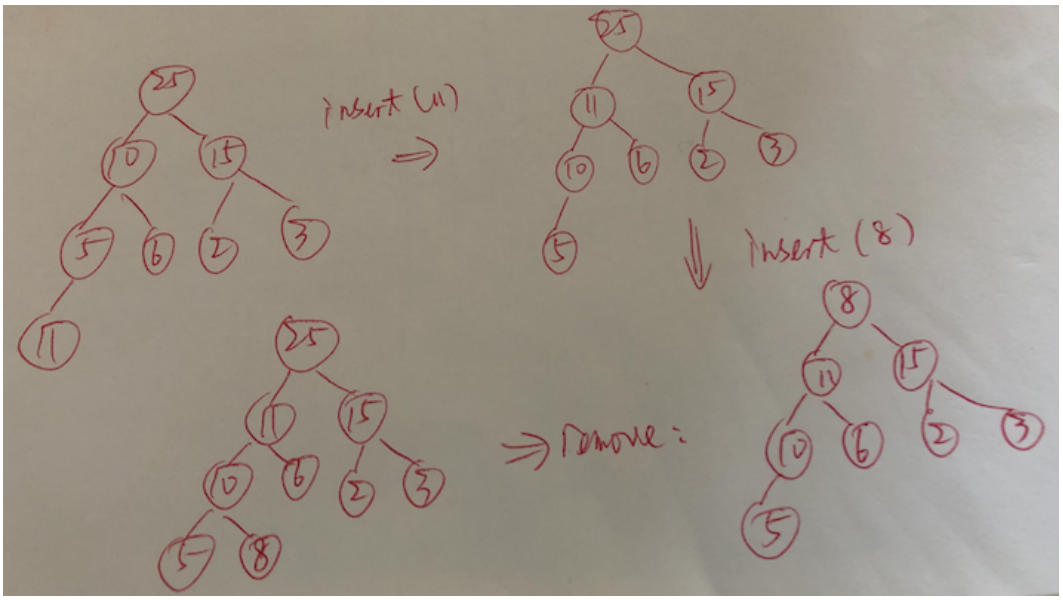


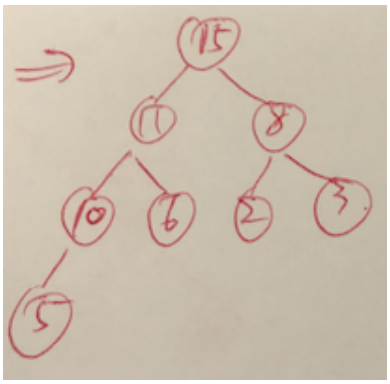


Left :is
Right: No. (It is not a complete binary tree)

[4pts] Consider the following heap. Draw the heap after these three successive operations:
insert(11)
insert(8)
remove()







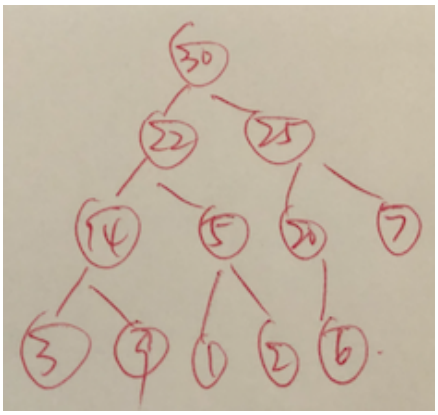
c). Given the following array representation of a binary heap:

30	22	25	14	5	20	7	3	9	1	2	6

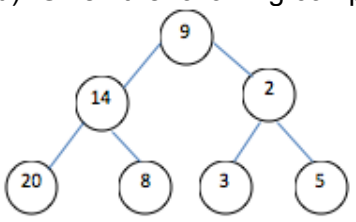
What is the left child of 20?_____6__

What is the right child of 14?____9___

What is the parent of 2?_____5____



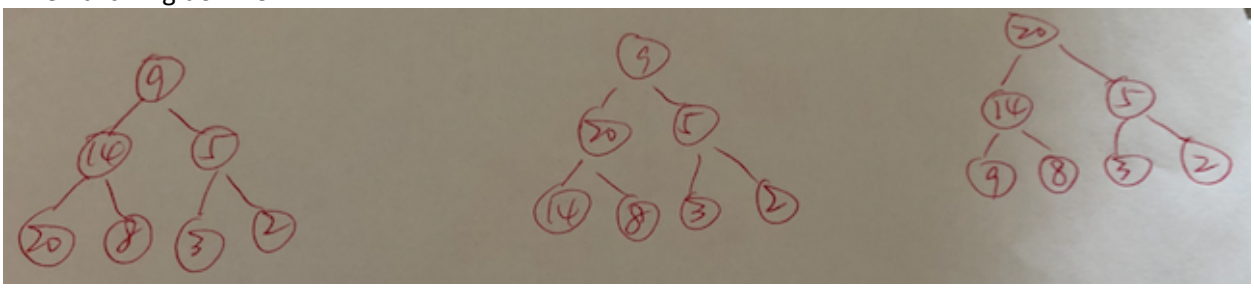
d). Given the following complete binary tree, draw the max-heap that can be constructed from it.



There are 7 nodes in the tree. So we can start at (n-1)/2 = 3 the node with index 3 to start the reheapification process.

```
for (int i=(n-1)/2; i>=0; i--){
    trickle_down(i)
}
```

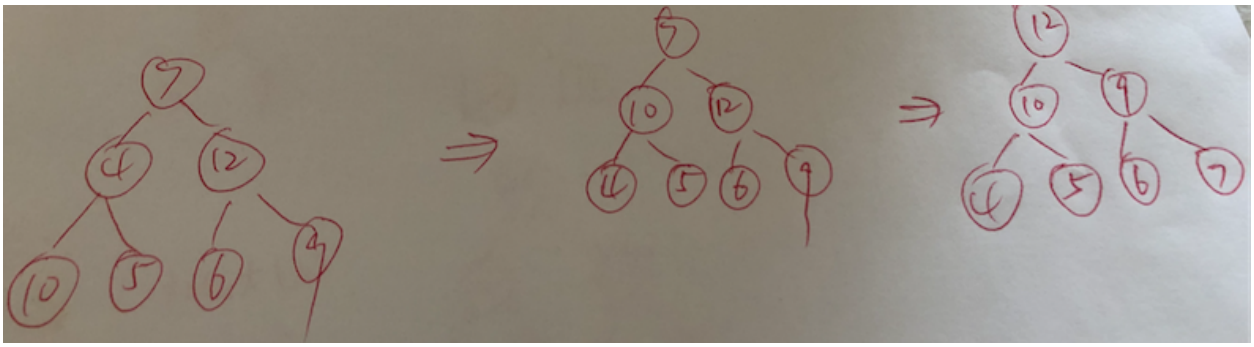
When trickling down 20, no change
When trickling down 2:
When trickling down 14:
When trickling down 9:



e). Suppose the elements of an array are in sorted order. Must it be a heap?

Yes.

f) Suppose we are sorting an array {7, 4, 12, 10, 5, 6, 9} using heap sort. Show the content of the array after the first iteration. You are recommended to show your working process (i.e. show the complete binary tree and max-heap constructed).



Problem 8. Hash Table

a). Consider linear probing, quadratic probing, and double hashing methods for collision resolution. Show the content of the hash table after inserting the keys listed below. Assume the hash functions is h(k) = k % 7. For the double hashing method, the second hash function (to determine the step size) is $h_2(k)$= (k % 3) + 2

Insert (from left to right): **13 27 35 17 24 38**

Linear Probing:

0	1	2	3	4	5	6
27	35		17	24	38	13

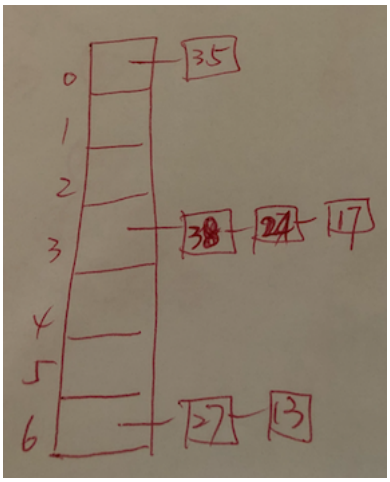
Quadratic Probing:

0	1	2	3	4	5	6
27	35		17	24	38	13

Double Hashing:

0	1	2	3	4	5	6
27	38		17	35	24	13

b). Consider using separate chaining methods for collision resolution. Show the content of the hash table after inserting the same list of keys in a): [13, 27, 35, 17, 24, 38] Assume the hash functions is also h(k) = k % 7.



c). Consider the following hash table, which uses linear probing to resolve collisions.

19	20		13	23	55	26	14	73	64

Give a sequence of insertions that would have caused the hash table to look like that.

One possible insert sequence is:

13 23 55 26 14 73 64 19 20

d). Try to write a HashTable application client (can either use the Java built-in Hashtable or HashMap class) to solve the following problem: Given a stream of inputting integers, find out and display each the element as well as its appearing frequency in the stream. You are supposed to solve this problem in O(N), where N is the number of integers in the stream.

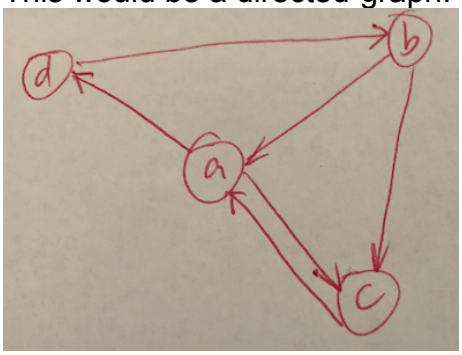
```
public void getFrequency(int[] arr){

// YOUR CODES

HashMap<Integer, Integer> map =  new HashMap<Integer, Integer>();

for (int i: arr) {

      if (!map.containsKey(arr[i]){

            map.put(arr[i], 1);

      } else {

          map.put(arr[i], map.get(arr[i])+1);

      }

}

 Set<Integer> set1 = map.keySet();

 for (int i: set1){

      System.out.println(i + " " + map.get(i));

 }

}
```

Problem 9. Graph

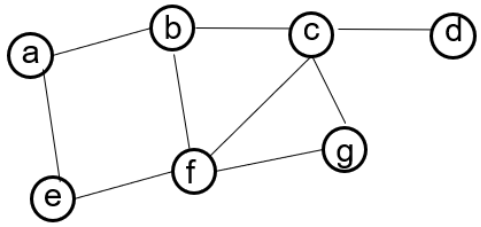
a). Given the following adjacency matrix, draw the corresponding directed graph.

	a	b	c	d
a	0	0	1	1
b	1	0	1	0
c	1	0	0	0
d	0	1	0	0

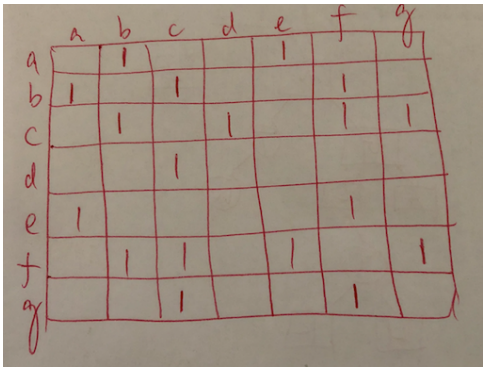
This would be a directed-graph. Here shows the graph.



b). In the following graph, we start searching from vertex **f** until we find vertex **d.** Among BFS and DFS, which algorithm finds vertex **d** faster? Show why. (faster means the algorithm visits less vertices until it finds the target). Also, note that you must use alphabetical order when choosing between multiple neighbors.



The adjacency matrix of this graph is:

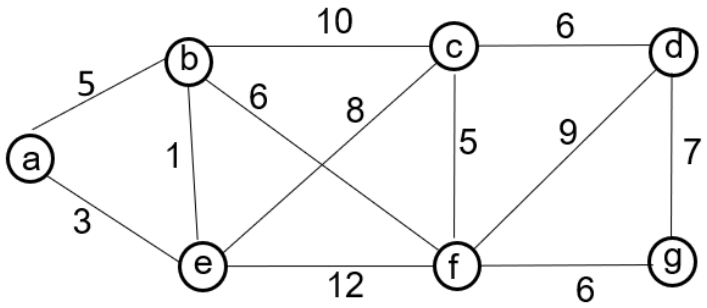


The BFS traversal sequence is: fbceadg

The DFS traversal sequence is: fbaecdg

They both find the node d is the same speed.

c). Show the steps involved in finding the shortest paths from the source node **a** using Dijkstra's algorithm. Initial state is given, please fill in the provided table.



Step	Visited Set(T)	Priority Queue	Selected node
1	{a}	(b,5) (e,3) (c,inf) (d,inf) (f,inf) (g,inf)	e
2	{a, e}	(b, 4), (c, 11), (d, inf), (f, 15), (g, inf)	b
3	{a, e, b}	(c, 11), (d, inf), (f, 10), (g, inf)	f
4	{a, e, b, f}	(c, 11), (d, 19), (g, 16)	c
5	{a, e, b, f, c}	(d, 17), (g, 16)	g
6	{a, e, b, f, c, g}	(d, 17)	d
7	{a, e, b, f, c, g, d}		

Previous node map
e -> a, b->a
b->e, c->e, f->e
f->b
d->f, g->f
d->c

Therefore, the shortest path from a to all other nodes are:

1) a->e->b: 4
2) a->e->c: 11
3) a->e->c->d: 17
4) a->e: 3
5) a->e->b->f: 10
6) a->e->b->f->g: 16

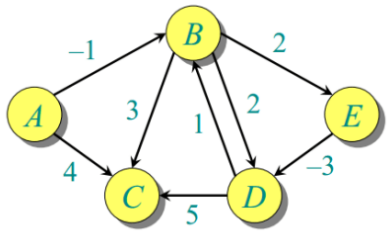
d). Apply the Kruskal's algorithm for finding minimum spanning tree on the graph above. Show the steps in the following table. Note that at every step you look at one edge, so in the first column write which edge is being considered.

Edge	Select?(Yes/ No)	Disjoint sets after selection decision
		{a}, {b}, {c}, {d}, {e}, {f}, {g}
b-e	yes	{a}, {b, e}, {c}, {d}, {f}, {g}
a-e	yes	{a, b, e}, {c}, {d}, {f}, {g}
a-b	no	{a, b, e}, {c}, {d}, {f}, {g}
c-f	yes	{a, b, e}, {c, f}, {d}, {g}
b-f	yes	{a, b, e, c, f}, {d}, {g}
c-d	yes	{a, b, e, c, f, d}, {g}
f-g	yes	{a, b, e, c, f, d, g}
d-g	no	
c-e	no	
d-f	no	
b-c	no	
e-f	no	

e). Show the steps involved in finding the Minimum Spanning Tree from the source node **a** using Prim's algorithm on the above graph.

Step	Visited nodes(T)	Edges on MST	Priority Queue	Selected edges
1	{a}	none	(a-b,5), (a-e:3)	a-e
2	{a, e}	(a-e:3)	(e-b: 1), (e-c: 8), (e-f: 12)	e-b
3	{a, e, b}	(a-e:3), (e-b:1)	(e-c: 8), (b-f: 6)	b-f
4	{a, e, b, f}	(a-e:3), (e-b:1), (b-f: 6)	(f-c: 5), (f-d: 9), (f-g: 6)	f-c
5	{a, e, b, f, c}	(a-e:3), (e-b:1), (b-f: 6), (f-c: 5)	(c-d: 6), (f-g: 6)	c-d
6	{a, e, b, f, c, d}	(a-e:3), (e-b:1), (b-f:6), (f-c:5), (c-d: 6)	(f-g: 6)	f-g
7	{a, e, b, f, c, g, d}	(a-e:3), (e-b:1),(b-f:6), (f-c:5), (c-d: 6), (f-g:6)		

f). Given the following graph, show the steps involved in finding shortest paths from the source node a to all other nodes using the Bellman-Ford algorithm. Please update the value in the following two arrays.



Since there are 5 vertices on graph, we need to repeat 5-1=4 iterations for the Bellman-Ford algorithm. In each iteration, we need to check every edge on the graph, and update the d[] and parent[] arrays accordingly.

Here follows show the d[] and parent[] array for each iteration.

d[] parent[]

	A	B	C	D	E		A	B	C	D	E
1	0	-1	2	-2	1			A	B	E	B
2	0	-1	2	-2	1			A	B	E	B
3	0	-1	2	-2	1			A	B	E	B
4	0	-1	2	-2	1			A	B	E	B

Problem 10. Others

a). What will be the value returned when calling the following recursive method by
foo(0, 6)

```
public static void foo(int low, int high) {
  if (low != high) {
     int mid1 = (low + high) / 2;
     foo(low, mid – 1);
     System.out.println(mid);
     foo(mid + 1, high);
  }
}
```

1
3
5

b) Which would be the best data structure for the following scenario: You want to keep a list of all tasks to be processed. You know the maximum number of tasks possible and you want to quick access to each task.
A. array
B. Stack
C. Queue
D. single-ended singly linked list.

A.

c) Which sorting algorithm would be the most efficient for an array already in ascending order.
A. Quick sort
B. Insertion sort
C. Merge Sort
D. Bubble Sort

B. For case when the array is already sorted, the time complexity for Insertion sort would be O(N). Others is O(nlogn).

d). Which algorithm can be applied to a directed graph that contains negative weights on edges but no negative cycles to find single-source shortest paths.
A. Dijkstra's Algorithm
B. Bellman-Form algorithm

B.
Dijkstra's algorithm cannot work on Graph with negative weights.